Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. Вып. 53. Автоматное программирование. 2008, с. 137–144.

УДК 004.4’242

ВЕРИФИКАЦИЯ ПРОГРАММ ПОСТОЕННЫХ НА ОСНОВЕ АВТОМАТНОГО ПОДХОДА С ИСПОЛЬЗОВАНИЕМ ПРОГРАММНОГО СРЕДСТВА *SMV*

Е. А. Курбацкий

В работе рассматривается метод верификации программ, построенных на основе автоматного подхода с использованием метода проверки на моделях (*Model Checking*). Для проверки модели используется программное средство *SMV*. При предлагаемом подходе система переходов не строится в явном виде, что позволяет верифицировать программы с большим числом состояний.

Введение

В работе [1] описан метод верификации программных систем, основанный на моделях (*Model Checking*). При использовании этого метода строится система переходов с конечным числом состояний. Свойства модели выражаются на языке темпоральной логики, после чего проверяется соответсвие модели свойствам. Одной из основных проблем методов *Model Checking* является необходимость построения конечной модели для проверки. Эта процедура может быть весьма сложной, а построенная в результате модель может обладать огромным числом состояний, что затрудняет верификацию. При попытке уменьшить число состояний путём абстрагирования от некоторых деталей реализации реальной программы возникает проблема адекватности полученной модели, а, следовательно, и проблема корректности результата верификации.

При использовании автоматного программирования [2] проблема, указанная выше, решается на этапе проектирования программы. В автоматной программе все состояния разделены на два класса [3]: управляющие и вычислительные. При этом управляющие состояния описываются набором конечных автоматов. Набор взаимодействующих автоматов уже является моделью, которая адекватна автоматной программе. Эта модель имеет конечное число состояний, что является необходимым условием для её верификации.

Данная проблема рассматривалась в ряде работ. В работе [4] описан способ проверки одного автомата. В работе [5] приводится способ проверки системы автоматов. При этом предлагается использовать программное средство *SPIN* [6].

Постановка задачи

Рассматривается задача проверки свойств системы автоматов, построенной на основе автоматного подхода [2]. Существует большое число программ реализующих верификацию модели. Поэтому можно не реализовывать алгоритмы для верификации непосредственно, а воспользоваться одним из существующих средств. В данной работе предлагается использовать программное средство *SMV* (*Symbolic Model Verifier*) [7]. Верификатор *SMV* предназначен для проверки того, что система переходов удовлетворяет требованиям, заданным на языке ветвящейся темпоральной логики C*TL*. В верификаторе *SMV* применяется описанный в работе [1] символьный алгоритм верификации моделей, основанный на упорядоченных двоичных диаграммах решений (*Ordered Binary Decision Diagram – OBDD).*

Необходимо построить программу, которой на вход поступает система автоматов и свойства на языке темпоральной логики, программа проверяет соответствие модели свойствам и возвращает контрпример, если свойства системы нарушены. В верификаторе *SMV*, для описания модели используется одноименный язык. Таким образом, задача сводится к следующим подзадачам:

- преобразовать программу в модель на языке *SMV*;
- преобразовать требования к системе в формулы темпоральной логики;
- запустить программу-верификатор *SMV;*
- преобразовать контрпример к модели в контрпример в автоматной программе.

На рис. 1 изображена схема предлагаемого подхода.

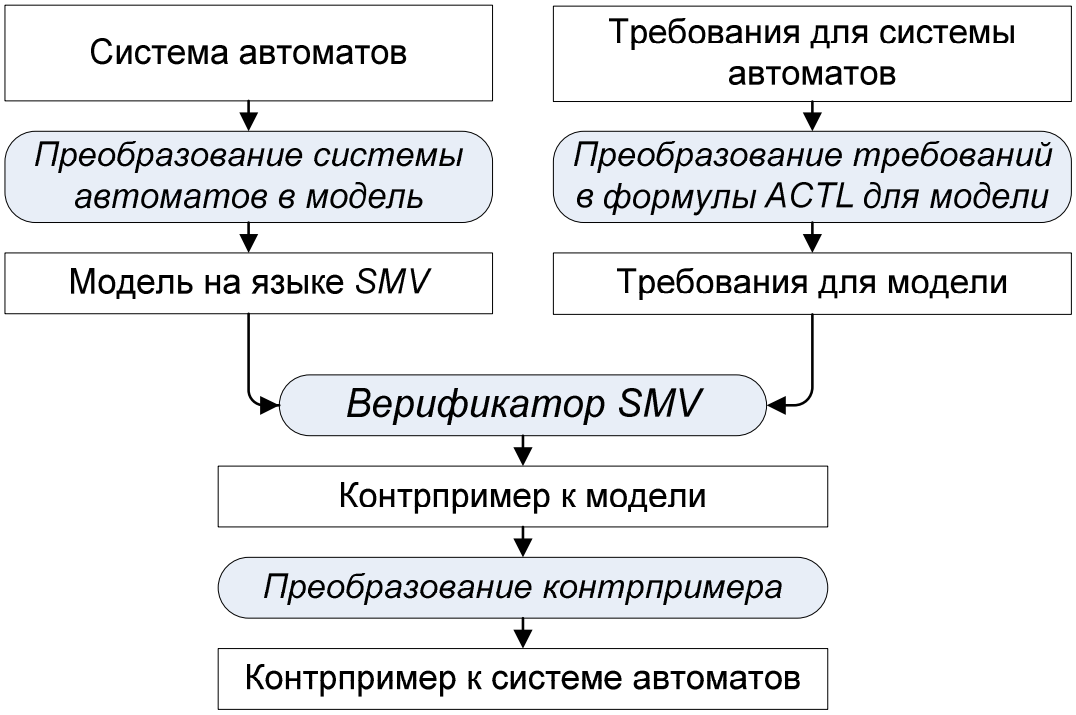


Рис. 1. Предлагаемый подход

Преобразование системы автоматов в модель

Построение модели выполняется в два этапа.

1. Так как система переходов допускает пометки только в вершинах, то для каждого автомата строится *модель автомата*. Вводятся дополнительные состояния на переходах, которые соответствуют действиям, выполняемым автоматом на переходах.
2. Полученные модели автомата объединяются в одну систему переходов и записываются как переменные и правила переходов между ними на языке *SMV*.

Рассмотрим задачу построения модели автомата. В автомате выделяются, помимо основных состояний, множество его *промежуточных состояний*, в которых автомат пребывает во время перехода из одного основного состояния в другое. В промежуточных состояниях будет храниться информация о том, какое действие автомат совершает. Промежуточное состояние автомата фиксируется каждый раз, когда автомат совершит одно из следующих действий:

- вызовет выходное воздействие, при этом в состоянии указывается, какое выходное воздействие вызывается;
- вызовет другой автомат, при этом в состоянии указывается, какой автомат и с каким событием вызывается;

Если никаких действий на переходе не выполняется, то выделяется одно дополнительное состояние, для того чтобы идентифицировать данный переход автомата.

Состояниями модели автомата будут состояния исходного автомата и *промежуточные состояния*. Условие и событие, записанные на переходе автомата записываются на переходе модели из основного состояния в промежуточное. Промежуточные состояния нумеруются так, чтобы начальное состояние автомата имело номер 0. На рис. 2 приведен пример выделения промежуточных состояний для одного перехода.

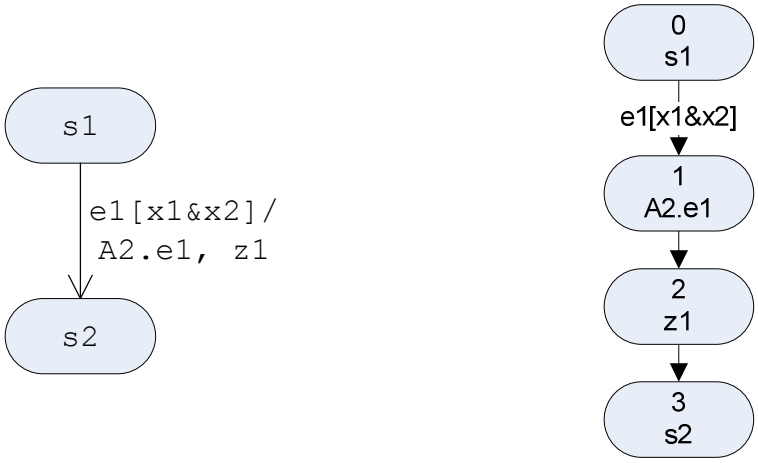


Рис. 2. Выделение промежуточных состояний на переходе.

Исходный переход (а), преобразованный переход (б)

Рассмотрим задачу преобразования нескольких моделей автоматов в систему переходов. Состояние модели будет описваться набором переменных. Каждому автомату A_k сопоставим переменную $State_k$. Для всей системы автоматов в целом введем переменные *Active, Event,* x_k $(0 \le k < m)$, где m – число входных переменных. Введенные переменные имеют следующий смысл:

- переменная $State_k$ содержит состояние модели, в котором может находиться каждая из n моделей. Каждая переменная может принимать значения от нуля до количества состояний модели.
- переменная *Active* содержит номер автомата, активного в данный момент или ноль, если никакой автомат не активен.
- переменная *Event* содержит имя события, которое передано автомату в данный момент. Событие может быть передано от внешнего источника или от другого автомата в результате вызова. Если в данный момент никакое событие не передается, то переменная *Event* принимает значение ноль.
- переменные x_1, x_2, x_m ... соответствуют входным воздействиям. Каждая переменная может принимать значения ноль (ложь) или один (истина).

Каждое состояние модели представляет собой объединение переменных $State_k$ $(1 \le k \le n)$, *Active*, *Event,* x_k $(0 \le k < m)$. Так как состояние задается набором переменных, то можно не строить систему переходов в явном виде. Таким образом, можно проверять модели с большим числом состояний.

Для того чтобы задать систему переходов, требуется выразить отношение между текущими и следующими значениями переменных. Зададим начальное значение всех переменных:

- переменные $State_k$ для всех k содержат состояния модели соответсвующие начальным состояниям всех автоматов;
- переменная *Active* – содержит ноль;
- переменная *Event* – содержит ноль;
- переменные x_1, x_2, ... x_m – содержат входные воздействия, которые могут принимать значения ноль или единица.

Теперь зададим правила переходов. Для $State_k$ запишем следующие правила переходов:

- если автомат активен *(Active = k)*, то изменение значения переменной $State_k$ выполняется в соответствии с переходами модели автомата *k*. Переменная $State_k$ может изменить свое значение на любое значение, в которое есть переход из текущего состояния в модели автомата, если условие, записанное на переходе, выполняется;
- если никакой переход невозможен, значение переменной $State_k$ не изменяется;
- если автомат не активен, то значение переменной остается прежним.

Опишем изменение значений переменной *Active*. Возможно несколько вариантов:

- если переменная *Active* равна нулю. Это означает, что никакой автомат в данный момент времени не активен, а, следовательно, требуется выбрать автомат, который будет выполняться. Этот выбор происходит недетерминировано из множества всех номеров автоматов. После того, как автомат для выполнения выбран, его номер записывается в переменную *Active*;
- иначе переменная *Active* равна *k*. Это значит, что активен автомат *k*. Для определения значения переменной *Active* необходимо рассмотреть следующее состояние модели *k*. Это состояние содержится в переменной $State_k$.
    - если состояние $State_k$ - промежуточное состояние, и в нем вызывается автомат *l*, следовательно, он будет активен на следующем шаге. Таким образом, переменной *Active* необходимо присвоить значение *l*;
    - если $State_k$ является состоянием исходного автомата, следовательно, автомат *k* уже закончил переход. Проверим, нет ли автомата *l*, который вызвал автомат *k:*
        - если автомат *l* был вызван каким-то другим автоматом *m*, то управление требуется вернуть к автомату *m*. Следующее значение переменной *Active* будет равно *m*;
        - иначе следующее значение переменной *Active* равно нулю.
    - иначе значение переменной *Active* не изменяется.

Переменная *Event* принимает следующие значения:

- если переменная *Active* равна *k*, и следующее значение переменной $State_k$ соответствует вызову автомата с событием *e*, то следующее значение переменной *Event* равно *e*;
- иначе следующее значение *Event* равно нулю.

Преобразование системы автоматов в модель для *SMV*

Опишем преобразование моделей автоматов в модель на *SMV*. Каждая модель автомата размещается в отельном модуле, с именем соответсвующим имени автомата. Каждый модуль будет иметь следующие параметры:

- `Active` – является ли данный модуль активным;
- `Event` – входящее событие, переданное автомату;
- Ap_1, Ap_2, ... – состояния экземпляров автоматов, с которыми данный модуль взаимодействует.
- x_1, x_2, ... – входные воздействия.

Определение модуля имеет следующий вид:

MODULE name(Active, Event, Ap_1, Ap_2, …, x_1, x_2, … x_m)

Каждый модуль будет хранить номер текущего состояния модели в переменной *State*. Она описывается State: 0..p;. Здесь p – число состояний модели данного автомата.

Далее требуется указать правила, по которым будут изменяться значения переменных. Эти правила описываются в секции ASSIGN.

В начале значение переменной *State* = 0. Это записывается как init(State) = 0; Для каждого состояния модели указывается, в какие состояния модель может переходить. Это делается при помощи ключевого слова TRANS, после него записывается предикат, истинность которого означает наличие перехода. Чтобы составить такой предикат запишем условие для каждого перехода, объединив их операцией логического или. Для каждого перехода из состояния s_i в состояние s_j, который происходит по событию e_{ij} при условии c_{ij}, записывается:

(Active & State = s_i & next(State) = s_j & Event = e_{ij} & c_{ij}) |

Для каждого состояния модели s_i запишем условие того, что ни одни переход из данного состояния не возможен.

(Active & State = s_i & next(State) = s_i & !(Event = e_{i1} & c_{i1})
& !(Event = e_{i2} & c_{i2}) & ...) |

Если автомат не активен, то состояние не изменяется.

(!Active & State = next(State))

Далее требуется выразить основные состояния автоматов. Это записывается в секции DEFINE:

DEFINE
s1 := State = n_1;
s2 := State = n_2;
. . .

Далее выразим свойство, что модель находится в состоянии автомата:

inState := State = n_1 | State = n_2 | …;

Здесь s1, s2, ... – имена состояний автомата, а n_1, n_2, ... – соответствующие им номера состояний модели.

Переменные, общие для всей системы, записываются в модуле main после ключевого слова VAR.

- переменные x_k описывающие входные воздействия, записываются как x_k: {0, 1};
- переменные, описывающие экземпляры автоматов записываются как *name*: *name*(Active = k, Event, Ap_1, Ap_2, …, x_1, x_2, … x_m), где *name* – имя автомата k, Ap_1, Ap_2 – имена автоматов, с которыми он взаимодействует, x_1, x_2, ..., x_m – входные воздействия;
- переменная Event описывается следующим образом:
  Event: {0, e_1, e_2, e_3, …};

Зададим значения переменных. В частности, начальное значение переменной *Active*.

init(Active) := 0;

Определим следующее значение переменной *Active*:

next(Active) := case

Если все автоматы не активны, то выберем любой автомат:

```
    Active = 0: 1..n;
```

Для всех состояний модели s_k и всех моделей A_k, в которых вызывается автомат:

```
    (Active = k & next(Ak.State) = sk): l;
    (Active != k & Ak.State = sk & Al.inState): k;
```

Для всех автоматов запишем условие, что он вернет управление, когда закончит переход:

```
    (Active = k & next(Ak.inState)): 0;
```

Иначе значение переменной *Active* не изменяется:

```
    1: Active;
  esac
```

Далее записывается описание переменной *Event*. Начальное значение переменной *Event:*

```
  init(Event) := 0;
  next(Event) := case
```

Для всех вызовов автоматов с событием e_k записывается.

```
    Active = k & next(Ak.State) = sk: ek;
```

Иначе значение переменной *Event = 0*:

```
    1: 0;
  esac;
```

Для каждого выходного воздействия z_k запишем:

```
DEFINE
  Zk = (Active = k1 & A1.State = s1) |
    (Active = k2 & A2.State = s2) | ...
```

Запись требований

Для записи требований используются формулы темпоральной логики *ACTL*. Опишем, как записываются свойства автоматной модели в виде формул *ACTL.*

Введем формулы, которые будут описывать состояния автоматов:

- для записи условия, что автомат A_k находится в состоянии s_j достаточно записать $A_k.s_j$;
- условие того, что выполнилось выходное воздействие z_1, записывается как z_1;
- для записи условия, что произошло событие e_i, достаточно записать Event = e_i;

Для записи формул, описывающих состояния автомата, также можно использовать логические операторы. Если f и g – формулы состояния, то формулами состояния являются:

- f & g – одновременно выполняются f и g;
- f | g – выполняется либо f либо g;
- f xor g – выполняется либо f либо g, но не одновременно;
- !f – не выполняется f;

- `f -> g` – если выполняется `f`, то выполняется `g`;
- `f <-> g` – тоже что и `(f -> g) & (g -> f)`.

Помимо свойств текущего состояния в условиях можно использовать темпоральные операторы: `AF`, `AF`, `AG`. Оператор `AX` не используются для записи свойств автоматов, так как в данной модели один переход автомата соответствует неопределенному числу переходов модели. Опишем эти операторы:

- `AF f` (*Future*) – оператор будущего. Означает, что на всех путях из текущего состояния существует состояние, когда формула `f` выполнится.
- `AG f` (*Global*) – оператор означает, что данная формула `f` будет выполняться на каждом пути из текущего состояния в каждом состоянии: `f` будет выполняться в каждом состоянии, достижимом из текущего состояния.
- `A[f U g]` (*Until*) – оператор истинен, только если на каждом пути когда-нибудь выполнится формула *g*, а до этого момента всегда будет выполняться формула `f`.

Преобразование контрпримера

Если опровергаемая формула принадлежит *ACTL*, то при её невыполнении получим контрпример в системе переходов. Любой контрпример для модели является либо конечным путем, либо путем с конечным началом и циклом.

Каждое состояние является набором переменных *Event, State$_k$, Active, x_k ($0 \le k < m$)*. Предлагается следующий алгоритм для преобразования контрпримера к системе переходов в контрпример к системе автоматов. Контрпример к системе автоматов также представляется в виде последовательности состояний, только вместо значений переменных информация предоставляется в терминах состояний и действий. Для каждого состояния контрпримера выведем следующую информацию о системе автоматов.

Если *Active = 0*, то никакой автомат не активен. Данное состояние не учитывается в контпримере для автомата.

Если *Active != 0*, то выводится название активного автомата. Также выводится действие активного автомата. Выводятся состояния всех автоматов, полученые из переменных *State$_k$*. Аналогично выводятся значения всех входных воздействий, записанных в переменные x_k, и значение переменной *Event*.

Пример построения модели системы автоматов

Продемонстрируем построение модели на небольшом примере. Рассмотрим два потока, которым необходимо время от времени работать с некоторым ресурсом. При этом необходимо, чтобы в каждый момент времени только один из потоков мог иметь доступ к ресурсу, но не оба сразу. Для моделирования этого примера, потребуются три автомата. Автомат *A0* - будет соответствовать ресурсу, а автоматы *W1* и *W2* - потокам. На рис. 3 изображена диаграмма переходов этого автомата.

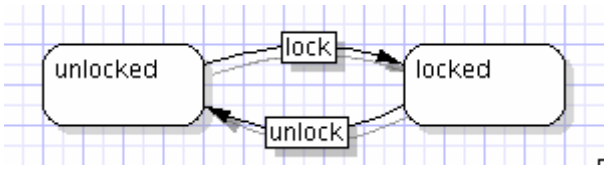


Рис. 3. Автомат A0

Автомат имеет два состояния:

- *unlocked* – начальное состояние означает, что ни один из потоков не использует ресурс. Автомат переходит из этого состояние в состояние *locked* по событию *lock*;
- *locked* – состояние означает что ресурс в данный момент используется. Автомат переходит из этого состояния по событию *unlock* в состояние *unlocked*.

На рис. 4 приведена диаграмма переходов автомата *W1*. Автомат *W2* устроен аналогично. Изначально оба автомата находятся в состоянии *wait.*

После чего один из них переходит в состояние *work*, в котором он может вызывать выходное воздействие *z1*. По окончанию работы по событию *e1* автомат возвращается в состояние *wait*.

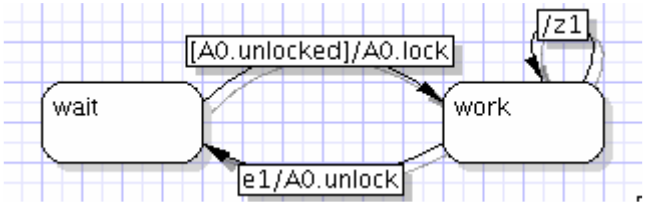
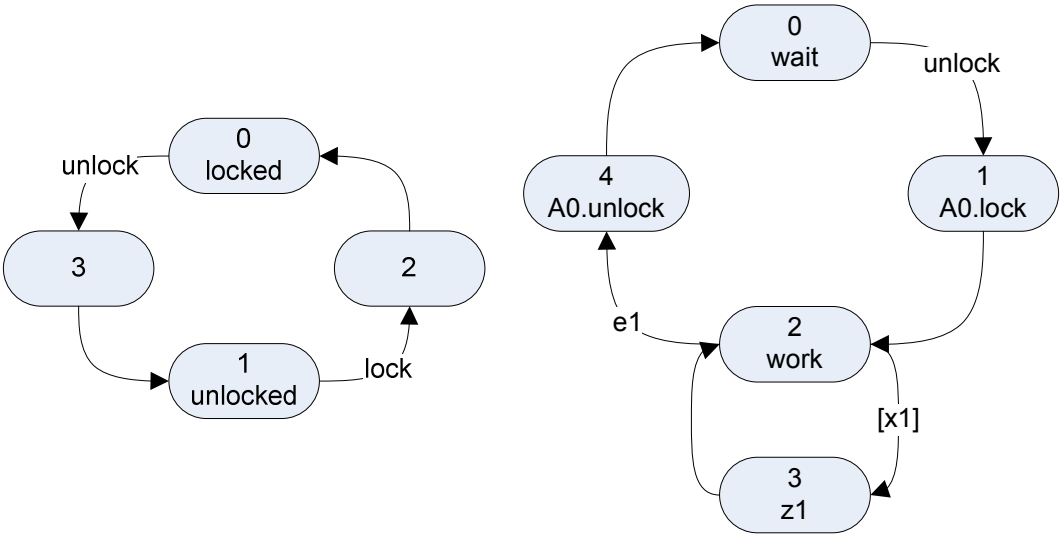


Рис. 4. автомата W1. (автомат W2 устроен аналогично)

На рис. 5 приведены модели автоматов A0 и W1.



а б

Рис. 5. Модели автоматов A0(а) и W1(б)

В приложении приведена модель на языке *SMV* для системы автоматов. Требуется проверить, что оба автомата не могут находиться в состоянии work одновременно. Это выражается формулой `AG(!(W1.work & W2.work))`. При проверке данной модели верификатор *SMV* возвращает, что модель соответствует формуле.

Литература

1. Symbolic Model Verifier. http://www.cs.cmu.edu/~modelcheck/smv.html
2. SPIN Model checker. http://spinroot.com/spin/whatispin.html
3. Шалыто А. А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука. 1998. http://is.ifmo.ru/books/switch/1

4. Вельдер С. Э., Шалыто А. А. О верификации простых автоматных программ на основе метода Model Checking //Информационно-управляющие системы. 2007. № 3, с.27–38. http://is.ifmo.ru/download/27-38.pdf
5. Кузьмин Е. В. Иерархическая модель автоматных программ // Моделирование и анализ информационных систем. 2006. № 1, с. 27–34.
6. Кларк Э., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. М.: МЦНМО. 2002.
7. Шалыто А. А., Туккель Н. И. Программирование с явным выделением состояний //Мир ПК. 2001. № 8, с. 116–121, № 9, с. 132–138.

Приложение

```
MODULE A0(Active, Event, x1)
VAR
  State: 0..3;
ASSIGN
  init(State) := 0;
TRANS
  (Active & State = 0 & next(State) = 1 & Event = lock) |
  (Active & State = 0 & next(State) = 0 & !(Event = lock)) |
  (Active & State = 1 & next(State) = 2) |
  (Active & State = 2 & next(State) = 3 & Event = unlock) |
  (Active & State = 2 & next(State) = 2 & !(Event = unlock))
                  |
  (Active & State = 3 & next(State) = 0) |
  (!Active & State = next(State))
DEFINE
  unlocked := (State = 0);
  locked := (State = 2);
  inState := (State = 0) | (State = 2);

MODULE W(Active, Event, x1, A0)
VAR
  State: 0..4;
ASSIGN
  init(State) := 0;
TRANS
  (Active & State = 0 & next(State) = 1 & A0.unlocked) |
  (Active & State = 0 & next(State) = 0 & !(A0.unlocked)) |
  (Active & State = 1 & next(State) = 2 & 1) |
  (Active & State = 2 & next(State) = 3 & x1) |
  (Active & State = 2 & next(State) = 4 & 1) |
  (Active & State = 4 & next(State) = 0 & 1) |
  (!Active & State = next(State))

DEFINE
  wait := (State = 0);
  work := (State = 2);
  inState := (State = 0 | State = 2);

MODULE main()
VAR
  x1: {0, 1};
```

```
  Event: {0, e1, lock, unlock};
  Active: 0..3;
  A0: A0(Active = 1, Event, x1);
  W1: W(Active = 2, Event, x1, A0);
  W2: W(Active = 3, Event, x1, A0);
ASSIGN
  init(Active) := 0;
  next(Active) := case
    Active = 0: 1..3;
    Active = 2 & next(W1.State) = 1: 1;
    Active = 3 & next(W2.State) = 1: 1;
    Active != 2 & W1.State = 1 & next(A0.inState): 2;
    Active != 3 & W2.State = 1 & next(A0.inState): 3;
    Active = 2 & next(W1.State) = 4: 1;
    Active = 3 & next(W2.State) = 4: 1;
    Active != 2 & W1.State = 4 & next(A0.inState): 2;
    Active != 3 & W2.State = 4 & next(A0.inState): 3;
    Active = 1 & next(A0.inState): 0;
    Active = 2 & next(W1.inState): 0;
    Active = 3 & next(W2.inState): 0;
    1: Active;
  esac;

  init(Event) := 0;
  next(Event) := case
    Active = 2 & next(W1.State) = 1: lock;
    Active = 3 & next(W2.State) = 1: lock;
    Active = 2 & next(W1.State) = 4: unlock;
    Active = 3 & next(W2.State) = 4: unlock;
    1: 0;
  esac;
SPEC AG(!(W1.work & W2.work))
```